R para iniciantes
Aula V Programação

Carlos Henrique Tonhatti

Universidade Estadual de Campinas

Dúvidas da última aula?

Sumário

Sumário

Como é feita a programação em R

Nas últimas aulas vimos:

- A sintaxe e a lógica das funções no R;
- Como organizar os dados nas classes de objetos do R;
- Como organizar passo-a-passo uma rotina no R (*script*).

Na aula de hoje vamos ver mais algumas técnicas que auxiliam na construção de uma rotina de trabalho no R.

Funções condicionais

Em muitas tarefas desejamos que o computador execute instruções diferentes dependendo de alguma condição lógica.

```
INÍCIO
   Abrir o Forno
   SE(Forno== acesso)
     Botar lenha
   CASO CONTRÁRIO
     Acender fogo
   Assar pão
FIM
```

```
AbraOForno()
   if(Forno==acesso)
{
   ColocarLenha()
} else {
   AcenderOForno()
}
Assar o pão
```

Funções condicionais — sintaxe

Sintaxe

```
# Executa apenas se condição== TRUE
if(condição){
    bloco de instruções
}

# Executa bloco 1 se condição== TRUE OU bloco 2 caso contrário.
if(condição){
    bloco 1
}else{
    bloco 2
}
```

A condição pode ser qualquer regra lógica.

Exemplos

```
#par ou impar?
a<-9
if(a%%2==0){
   print("É par")
}else{
   print("É impar")
}
 [1] "É impar"
```

ifelse

Condicional mais simples

# Retorna um vetor com os valores indicados
ifelse(condição,valor se verdadeiro, valor se falso)

```
numero<- c(0,1,2,3,4,5,6,7,8)
ifelse(numero%%2==0,"par","impar")
 [1] "par"  "impar"  "par"  "impar"  "par"  "impar"  "par"
 [8]  "impar"  "par"
```

Funções de repetição

Em algumas tarefas desejamos que um bloco de instruções repetidamente até que uma condição seja satisfeita.

```
INÍCIO
Turma <- NotasAlunos
CalcularMedia(Aluno1)
CalcularMedia(Aluno2)
CalcularMedia(Aluno3)
...
FIM
```

```
INÍCIO
Turma<-NotasAlunos
Repita i de 1 até Numero de alunos
    CalcularMedia(Alunoi)
FIM
```

Funções de repetição

Funções de repetição

# Para cada repetição o objeto *i* recebe um valor da sequência. Termina no último elemento da sequência.
for(i in sequência){
bloco de instruções
}

# Testa se a condição for verdadeira executa o bloco de instruções. Repete até a condição se tornar falsa.
while(condição){
bloco de instruções
}

Exemplo

```
alunos<-matrix(1:12,ncol=3)
numero.alunos<-dim(alunos)[1]
medias<-c()
for(i in 1:numero.alunos){
    medias[i]<-mean(alunos[i,])
}
```

Neste caso poderia usar a função `apply(alunos,1,mean)`

Exemplo

Saber se um número é primo

```
numero<-13
aux<-numero-1
primo<-TRUE
while(primo==TRUE & aux>1){
       if(numero%%aux==0){
            primo<-FALSE}
       aux=aux-1
} print(primo)
```

Cuidado com repetições

Função que repetem instruções *"loops"* podem trazer uma série de problemas, entre eles:

- "*loops* infinitos" a repetição nunca para. A condição lógica sempre é verdadeira;
- "Nunca acontece". A condição lógica nunca é verdadeira;
- "Big loops". Quantidade de dados muito grande. Demora muito pra rodar.

Soluções:

- Verifique atentamente as condições lógicas;
- Tente outras funções: apply, tapply, foreach ...[1]

[1] Assunto da próxima aula.

Definição

Função

É uma sequência de instruções que fazem uma tarefa especifica empacotada como uma unidade.

Dependendo da linguagem pode receber outros nomes: procedimento, rotina, subrotina, subprograma, método.

Classe `function` no R

No R funções são objetos da classe `function`.

```
> class(plot)
 [1] "function"
```

Podemos criar funções no R de modo parecido como criamos outros objetos.

Criando uma função

Sintaxe da função

```
nome<-function(argumentos da função)
                {
                 Instruções
                 return()
                }
```

Exemplo

```
numero<-13
aux<-numero-1
primo<-TRUE
while(primo==TRUE & aux>1){
      if(numero%%aux==0){
            primo<-FALSE}
      aux=aux-1
}
print(primo)
```

```
>Ehprimo<-function(numero){
aux<-numero-1
primo<-TRUE
while(primo==TRUE & aux>1){
      if(numero%%aux==0){
            primo<-FALSE}
      aux=aux-1
} return(primo)
}

> Ehprimo(13)
 [1] TRUE
```

Exemplo

```
media <-function(x)
      {
       soma=sum(x)
       nobs=length(x)
       media=soma/nobs
       return(media)
      }
```

Uma função é um objeto

Como qualquer objeto criado uma função é listada no espaço de trabalho.

```
>ls()
 [1] Ehprimo media ...
```

Da classe `function`

```
> class(Ehprimo)
 [1] "function"
```

Escopo dos objetos de uma função

Objetos criados dentro de uma função só existem dentro da mesma função.

```
> media <-function(x)
      {
      soma=sum(x)
      nobs=length(x)
      media=soma/nobs
      return(media)
     }
> media(c(3,4,5))
 [1] 4
> soma
Erro:  objeto 'soma' não
encontrado
```

```
> mediaEsoma <-function(x)
      {
      soma=sum(x)
      nobs=length(x)
      media=soma/nobs
      return(c(media,soma))
     }
> mediaEsoma(c(3,4,5))
 [1] 4 12
# Não criou o objeto "soma"
apenas retornou o valor.
```

Escopo dos objetos de uma função

Objetos criados fora de uma função estão disponíveis para uma função:

```
> soma<-sum(c(3,4,5))
>media2 <-function(x)
       {
       nobs=length(x)
       media=soma/nobs
       return(media)
      }
media2(c(3,4,5))
 [1] 4
```

Cuidado! Quando criar uma função sempre declare os argumentos entre (). Esperar que um objeto exista fora da função é fonte de problemas.

Sumário

Erros no programa

Encontrar um erro em um programa é algo bem comum. Nem sempre o programa emite um sinal de alerta. O usuário precisa estar atento para detectar e corrigir erros.

Depuração do código

Do inglês “*Debugging*”,

- Remover impurezas, sujidade ou imperfeições; tornar(-se) puro. = LIMPAR, PURIFICAR
- Limpar(-se) de falhar moral. = EXPURGAR, PURIFICAR
- Tornar(-se) melhor em alguma coisa. = APERFEIÇOAR, APRIMORAR, APURAR
- Detectar e eliminar erros em programa de computador

.

in Dicionário Priberam da Língua Portuguesa [em linha], 2008-2013, http://www.priberam.pt/dlpo/depurar [consultado em 17-07-2014].

Princípios de depuração

Beware of bugs in the above code, I have only proved it correct, not tried it.

Donald Knuth

1. Confirme
2. Comece pequeno
3. Depure de forma modular

In: Norman Matloff. The Art of R programming, No Starch press, 2011

Confirmação

Confirme *um-por-um* cada aspecto que você considera verdade sobre o código se *é realmente* verdade.

```
> a<- 10
> b<-20
> a<-b
> b<-a
> b
 [1] 20
```

Começar pequeno

Quando iniciar o processo de depuração faça testes pequenos com casos simples. Teste cada função ou trecho do código.
Depois teste com conjuntos de dados maiores e vários funções ao mesmo tempo.

Depure de forma modular

- Escreva o código em forma de módulos e depois depure cada módulo.
- Faça funções que executem cada módulo, teste cada uma em separado.
- Faça um código principal curto com apenas os passos principais, deixe as funções que você criou em outro arquivo.

```
source(funcoes.r)
dados<-read.table(...)
permutacao<-teste.permutacao(dados)
chi<- testdochi(dados)
resultado<-list(permutacao,chi)
write.table(...)
```

Ferramentas de depuração

O R possui algumas ferramentas que auxiliam na correção e detecção de erros.

Pare se não

# Para a execução se a condição não for satisfeita
stopifnot(condição)

```
for(i in 1:10){
      cat(i)
      stopifnot(i>5)
}
1
Erro:  i > 5 is not TRUE
```

Ferramentas de depuração

Ferramenta `debug`

```
# Marca uma função para ser depurada.
debug(função)

# Desmarca uma função da depuração
undebug(função)
```

Toda vez que se executa uma função que está marcada para ser depurada é feita a depuração linha por linha da função.

Exemplo

Usando a função `media` criada na aula anterior.

```
> debug(media)
> media(c(1,2,3))
debugging in:  media(c(1, 2, 3))
debug em #2:  {
soma = sum(x)
nobs = length(x)
media = soma/nobs
return(media)
}
```

Exemplo (cont.)

```
Browse[2]>
debug em #3:  soma = sum(x)
Browse[2]>
debug em #4:  nobs = length(x)
Browse[2]>
debug em #5:  media =
soma/nobs
Browse[2]>
debug em #6:  return(media)
Browse[2]>
exiting from:  media(c(1, 2,
3))
 [1] 2
undebug(media)
```

- Executa linha por linha da função;
- Possibilita ver cada objeto intermediário;
- Digite “Q” para sair antes do final;
- Não esqueça de desmarcar.

Ferramentas de depuração

Ferramenta `browser`

```
# Executa o debug uma única vez a partir de uma linha específica
funcao<-function(argumentos){
        ...
         browser()
        ...
        return( ... )
}
```

Exemplo

```
> media <-function(x)
    {
        soma=sum(x)
        nobs=length(x)
        browser()
        media=soma/nobs
        return(media)
    } > media(c(1,2,3))
Called from:  media(c(1, 2, 3))
Browse[1]>
debug em #6:  media = soma/nobs
Browse[2]>
debug em #7:  return(media)
Browse[2]>
 [1] 2
```

Sumário

Definição

Otimização de programa

É um processo de modificação de um programa para que ele trabalhe de forma mais eficiente ou com menos recursos.

Um programa pode ser otimizado para que ele seja mais rápido ou capaz de operar com menos memória ou outros recursos. Aumentando a eficiência do programa.

Avisos sobre otimização

“Premature optimization is the root of all evil.”

Donald Knuth

- Você precisa otimizar?
- Um código apenas 2 ou 3 vezes mais rápido pode não compensar o trabalho;
- Um código que seja 10 vezes mais rápido pode compensar;
- Antes de otimizar você precisa ter um código que funcione!

Otimizando um código

Evite a `repetição`

Comandos de repetição (principalmente o `for` gastam muito tempo para serem executados. Evite-os!. Tente usar outras abordagens:

- Funções `apply, tapply,aggregate, etc`
- Lembre-se que operações com vetor são feitas elemento por elemento. Não é necessário `for()` em muitas situações.

```
>for(i in 1:length(numeros)){
   aux[i]<-numeros[i]/2
}

# Versão mais rápida
aux<-numeros/2
```

Otimizando o código

Escrevendo funções

- Escreva uma função que faça apenas o que você quer;
- Faça uma versão compilada da sua função ex: `cmpfun(função)`
- Escreva funções pequenas e importantes em `C` e use elas pelo `R`;

Otimizando o código

Preparando os dados

- Prefira usar matrizes ao invés de data frame;
- Limpe os dados antes e então use funções mais simples ex: `sum(x)/length(x)` é mais rápido que `mean(x)` pois não tem verificação de erros.
- Se possível, divida os dados em conjuntos menores, analise separadamente e depois junte as respostas *Computação paralela*. Vários pacotes ajudam a fazer isso.

Medindo o tempo

Com a função `system.time()` é possível saber quando tempo demora pra executar uma função. Use para comparar entre duas funções.

```
>system.time(Ehprimo(4001))
 usuário   sistema   decorrido
   0.012     0.000       0.012
> system.time(Ehprimo(400001))
 usuário   sistema   decorrido
   1.133     0.000       1.134
```